# **AAC Dashboard Readme**

## About the Project/Project Title

AAC Dashboard is an implementation of CRUD (Create, Read, Update, Delete) functionality on the AAC (Austin Animal Center) database, and an associated Dash GUI for dynamically filtering and displaying the retrieved data.

## Motivation

The goal of this application is to enable users to effectively interact with the Austin Animal Center Outcomes database. It was created for a client, international rescue-animal training company Grazioso Salvare, to help them identify dogs that are good candidates for search-and-rescue training. At the client's request, all code for this project is open-source, so that it may be used and adapted by similar organizations.

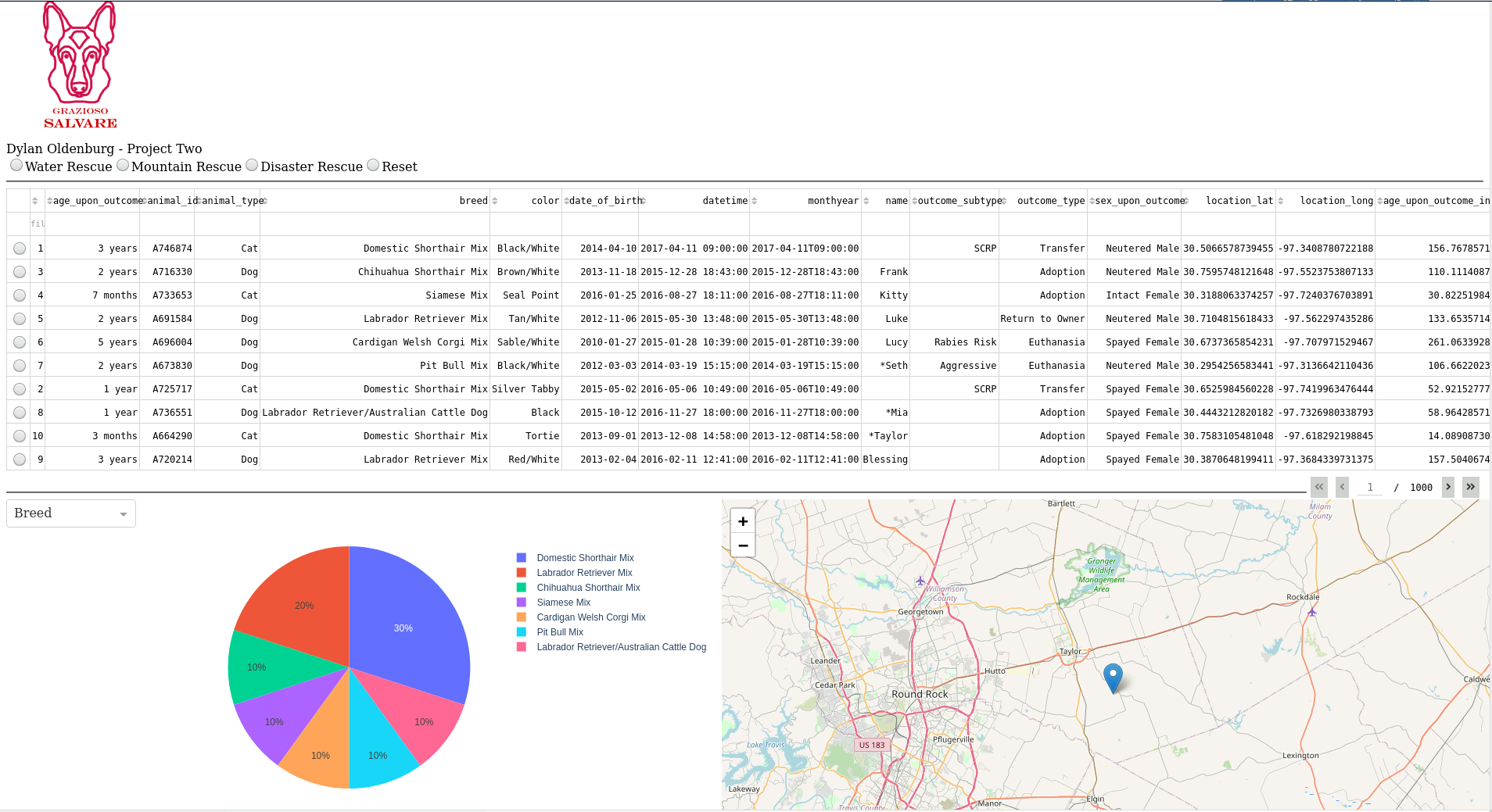
**Required Functionality**

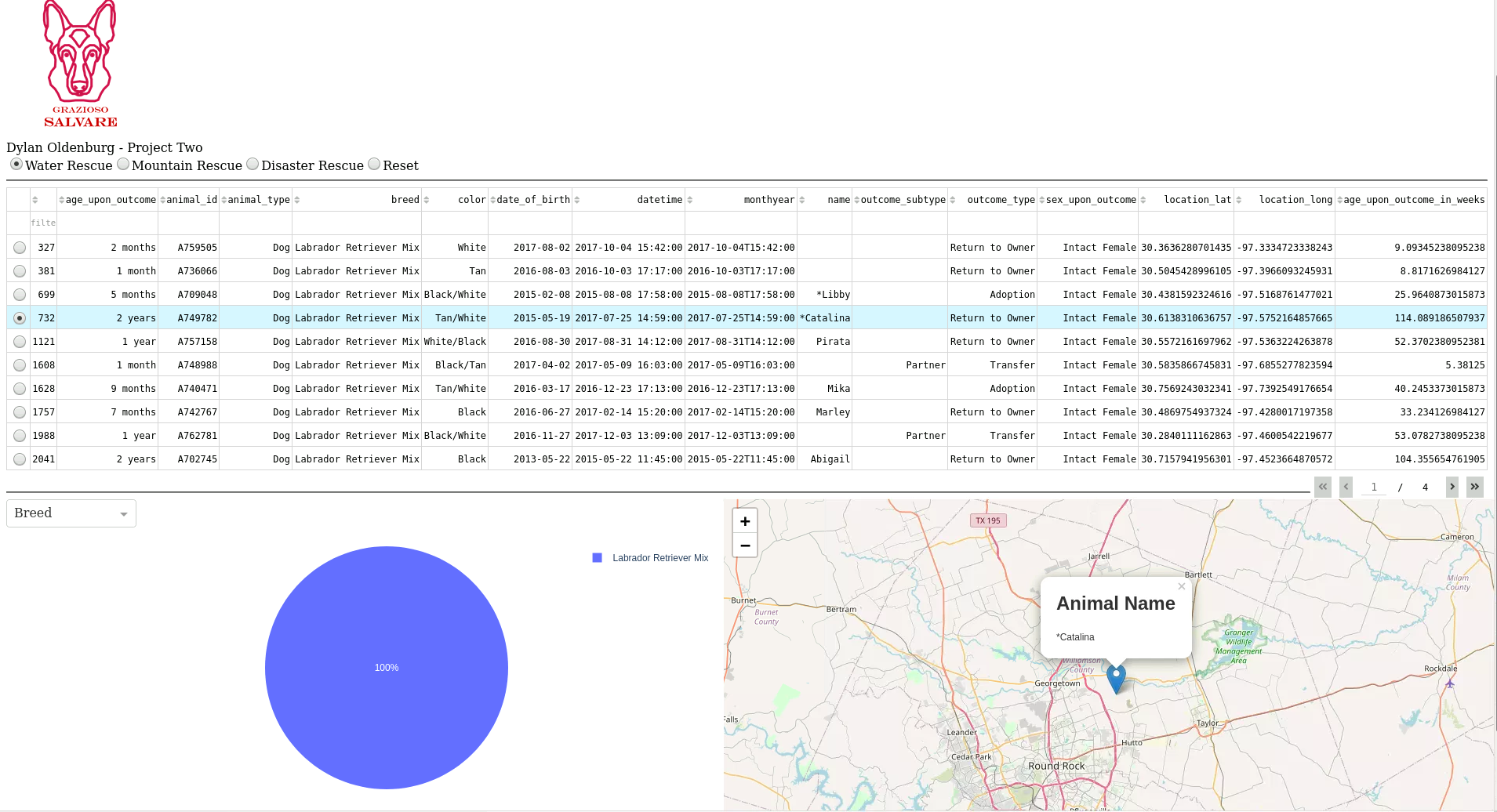
The core functionality required and implemented by this project is as follows:

* Access, filter, and display data from the AAC database. Specifically, functionality should exist to filter the data based on three client-specified criteria:
  + Animals suitable for Water Rescue training
  + Animals suitable for Mountain Rescue training
  + Animals suitable for Disaster Rescue or Individual Search training
* In addition to displaying the raw data in a table, display graphical representations of the data in the form of a graph and a geolocation map.

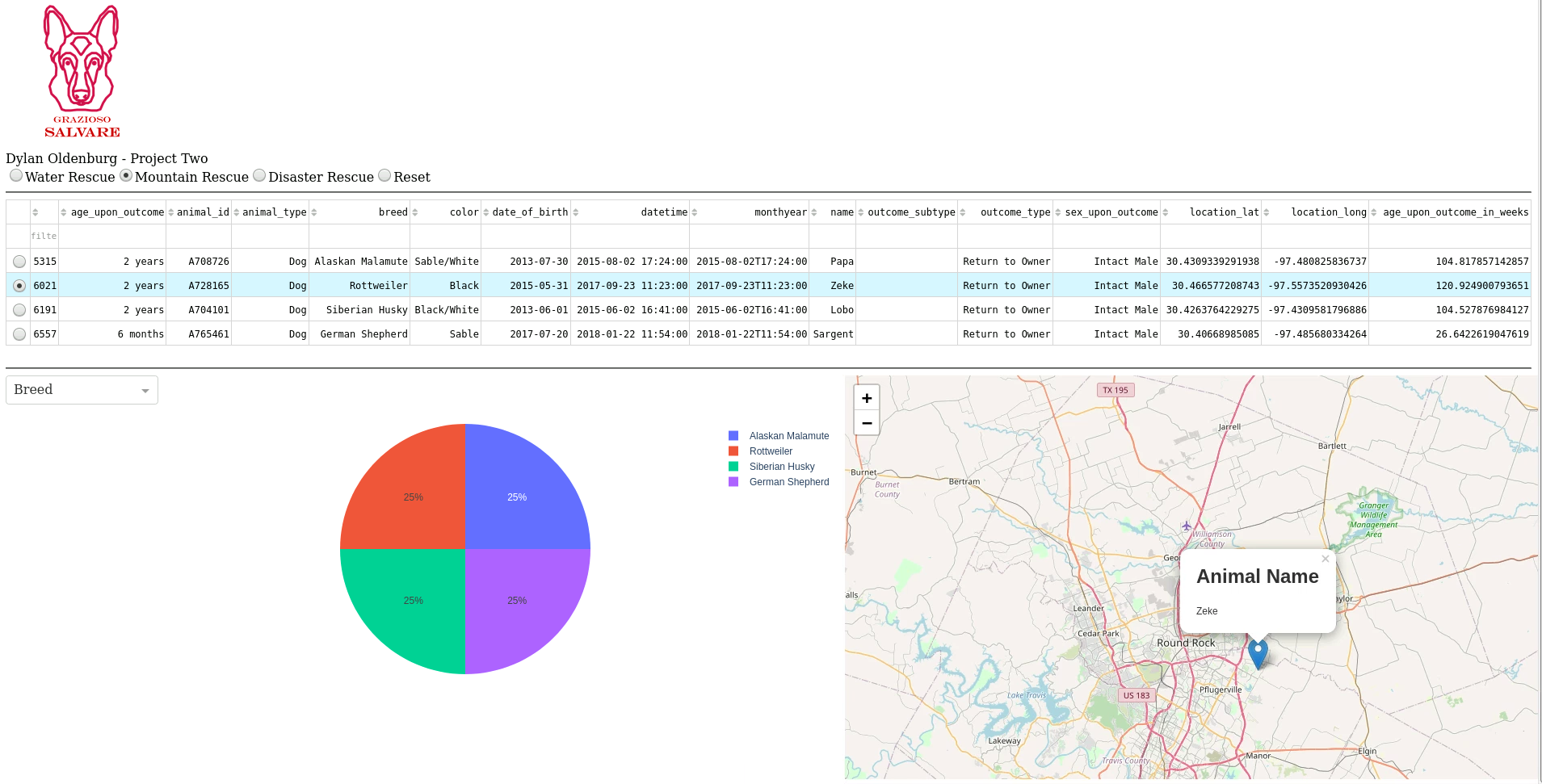
Implementation of this required functionality is demonstrated in the following screen captures:

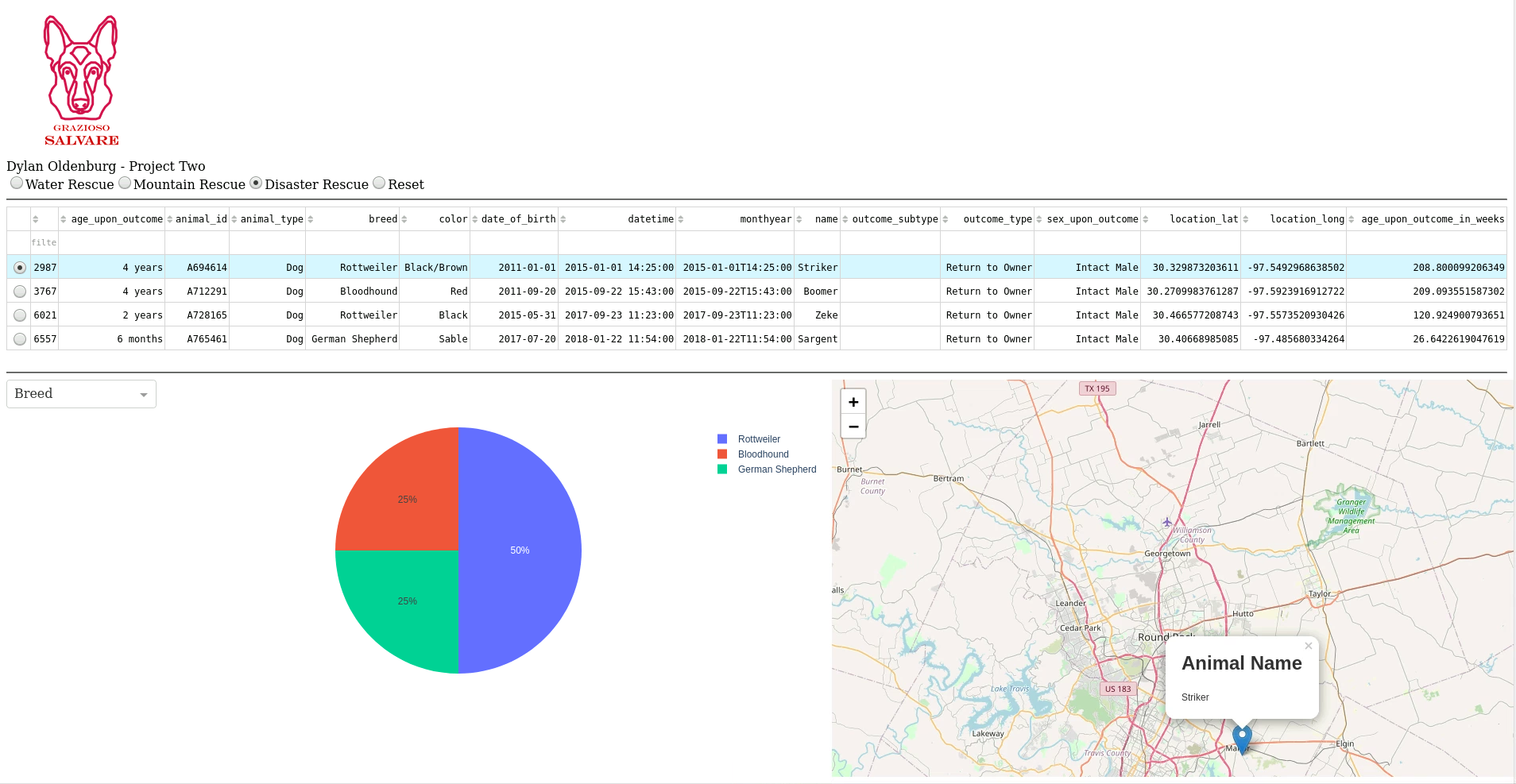
***Starting state of the dashboard, with no filters applied:***

***Dashboard filtered for Water Rescue candidates:***

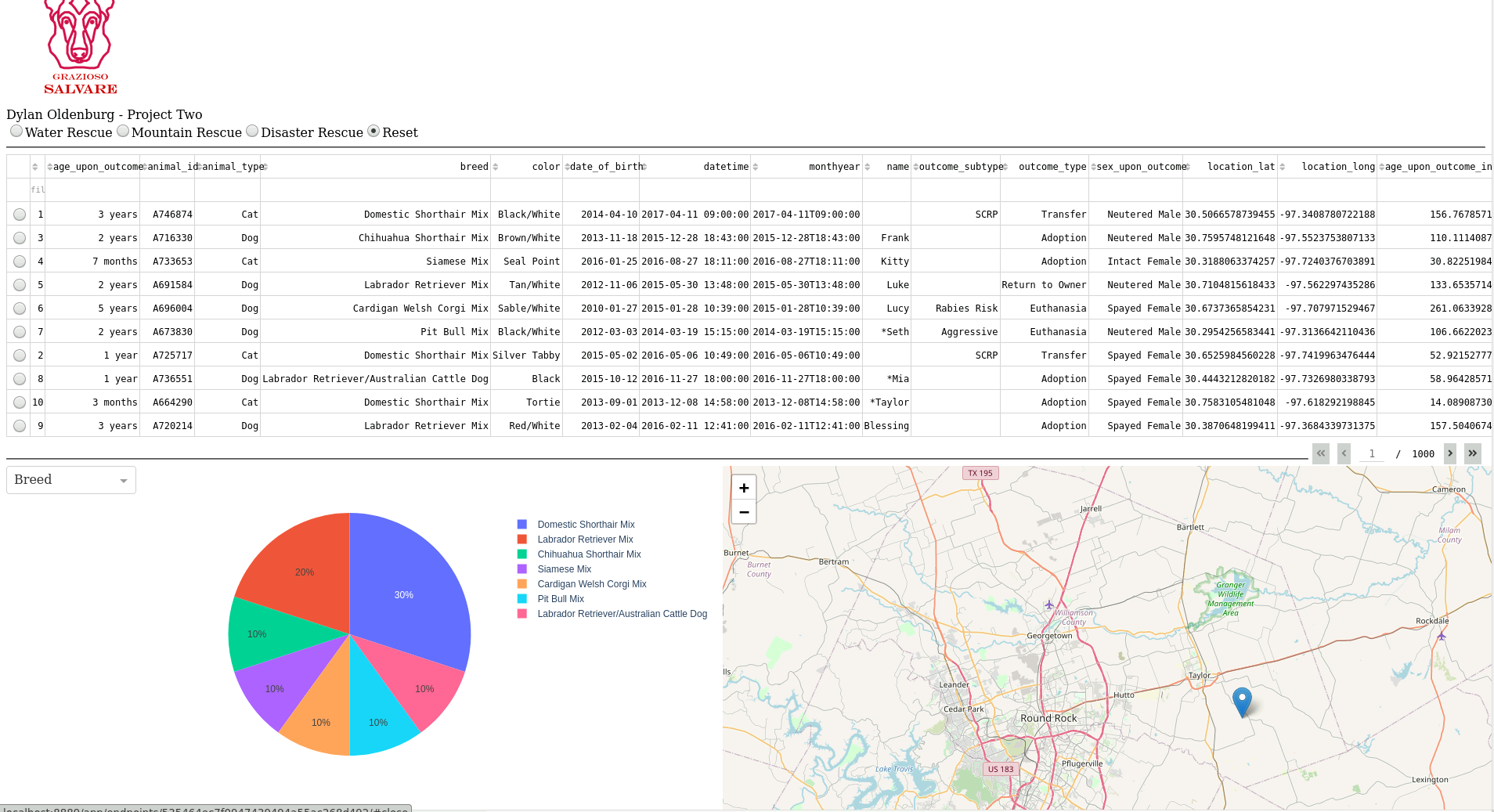
****

***Dashboard filtered for Mountain Rescue candidates:***

***Dashboard filtered for Disaster Rescue candidates:***



***Dashboard filters cleared to default values:***



**Tools Used:**

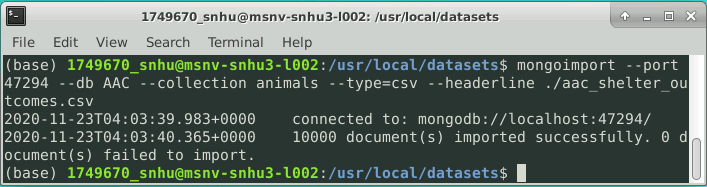
AAC Dashboard was developed using the MVC (Model-View-Controller) design pattern. To achieve the functionality described and pictured above, the following tools were used:

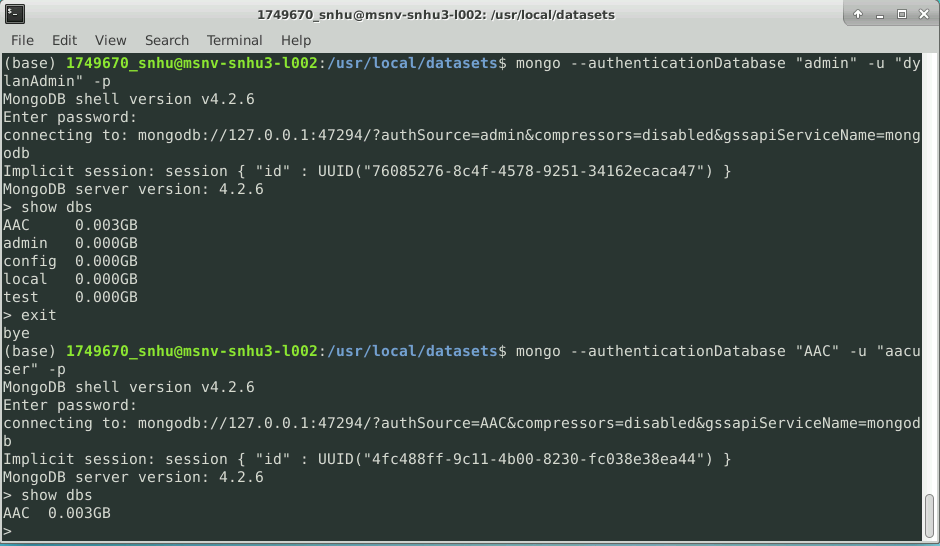
* **MongoDB:** Representing the Model component of the MVC pattern, MongoDB is a popular modern "NoSQL" database program. MongoDB is a highly versatile general purpose NoSQL database, combining a full range of SQL equivalent operators with support for nested document storage via JSON objects. This makes MongoDB an ideal choice for many projects. Installation instructions for MongoDB can be found here:
  + Windows: https://docs.mongodb.com/manual/tutorial/install-mongodb-on-windows/
  + Linux: https://docs.mongodb.com/manual/administration/install-on-linux/
  + macOS: https://docs.mongodb.com/manual/tutorial/install-mongodb-on-os-x/
* **PyMongo:** PyMongo is a Python distribution containing tools for working with MongoDB. It enables scripting and execution of MongoDB commands and queries using Python. It was chosen for this project because it is the officially-recommended way to work with MongoDB from Python, as stated on MongoDB.com. Information about PyMongo, including installation instructions, can be found here:
  + https://docs.mongodb.com/drivers/pymongo
  + https://pymongo.readthedocs.io/en/stable/
* **Dash:** Representing both the View and Controller components of the MVC pattern, Dash is an open-source Python framework used to build web applications for data visualization and analysis. Dash is fully browser-based, and facilitates rapid and easy construction of complete GUIs and web applications. Information about Dash, including installation instructions, can be found here:
  + https://dash.plotly.com/installation
* **Jupyter Notebook:** From Jupyter.org: "Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text." This application uses Jupyter to create, edit, and share Python scripts for interacting with the database. Both the Dash dashboard and the backend PyMongo CRUD module were developed and tested using Jupyter Notebook. Installation instructions for Jupyter Notebook can be found here:
  + https://jupyter.org/install

**Steps Taken to Complete Project:**

As a full stack application, this project implements a backend database and a user-facing front-end GUI to access that database. A Python module was developed as a "middleware" layer to facilitate communication between the front-end and back-end components. The first step was implementing the required database functionality using MongoDB. This included implementing user authentication for both admin and user level accounts, and importing the AAC's .CSV file into MongoDB. This brought the database online to be accessed by the Python middleware.

***Importing the dataset into MongoDB:***

***Authentication for both admin and user accounts:***

******

Once the database was successfully implemented, a Python module was developed to enable Python applications to execute MongoDB commands. This Python module defines a class called AnimalShelter which contains methods for the CRUD data access operations. The AnimalShelter class defines five methods: create(), readOne(), readAll(), update(), and delete():

* create() accepts an argument "data" which must be a dictionary. The supplied data will be added to the database. The method returns either True or False, corresponding to the success or failure of the create operation.
* readOne() accepts an argument "data" which must be a dictionary. The supplied data will be used to execute a MongoDB findOne() query. The method returns a list of all database entries matching the query. *Note that the readOne() method is not utilized by the AAC Dashboard, but is included in the project's CRUD functionality for potential future utility.*
* readAll() accepts an argument "data" which must be a dictionary. The supplied data will be used to execute a MongoDB find() query. The method returns a list of all database entries matching the query.
* update() accepts an argument "data" which must be a dictionary, as well as strings representing the key/value pair to be updated. The supplied data will be used to execute a MongoDB update\_many() query, and all database entries matching the query will be updated with the supplied key/value pair. The method returns the operation result in JSON format.
* delete() accepts an argument "data" which must be a dictionary. The supplied data will be used to execute a MongoDB delete\_many() query, and all data entries matching the query will be deleted from the database. The method returns the operation result in JSON format.

Once the Python middleware was developed, the final step was to implement the user-facing dashboard, which was developed using the Dash framework. This involves a combination of HTML and Dash components to define the layout and content of the web application. The dashboard imports and instantiates the CRUD class AnimalShelter, which it uses to log into the MongoDB database. A dataframe is displayed, populated initially by a readAll({}) query, returning the entire contents of the database. Radio buttons are implemented to enable the user to select from pre-defined filters, which update the dataframe to contain only the data entries that match the chosen filter. A pie chart was created using Plotly Express to provide visual representation of the currently-displayed data. Finally, a geolocation map was implemented, dynamically displaying the location of the currently-selected animal, along with information about the animal's name and breed.

**Challenges Encountered in Development:**

AAC Dashboard was developed in an academic environment over the course of eight weeks by a single developer: me! Because this was my first exposure to most of the tools used in its development (MongoDB and Dash in particular), I encountered many challenges and difficulties along the way. Thankfully, I had the benefit of both thoroughly-documented frameworks, the assistance of my course professor, and a community of classmates to share issues and ideas with, all of which were indispensable in meeting and defeating the various challenges I encountered. A few of those challenges include:

* Core CRUD functionality in the middleware Python module -- Initially, I struggled to correctly read from the database with my Python module. The cause of this issue was a missing return statement in my read() method, an error I found after discussion with my professor and a classmate who was having the same problem.
* Implementation of authentication in MongoDB -- Initial attempts to implement authentication in my Python middleware module failed. I ultimately determined that the cause was improper assignment of the "authSource" parameter in the client declaration. The following documentation was critical in resolving this obstacle:
  + https://pymongo.readthedocs.io/en/stable/examples/authentication.html
* Dynamically updating the geolocation map in the Dash dashboard -- The geolocation map was initially hardcoded with position values and row/column indices for data display. To fulfill the project requirements, it was necessary for these properties to be dynamic based on user selections. My first attempts at dynamically updating the geolocation map resulted in a broken callback that prevented the map from displaying at all. I resolved this issue with the help of some advice from a classmate, who provided links to the following vital documentation and resources:
  + https://dash.plotly.com/datatable/interactivity
  + https://www.shanelynn.ie/select-pandas-dataframe-rows-and-columns-using-iloc-loc-and-ix/
* Proper formatting of Dash dashboard elements -- the core project requirements called for a graph (I chose a pie chart), and a geolocation map to be displayed side-by-side. In addition to these, I wanted to implement a dropdown menu so that the user could modify the database column used by the pie chart. In my first attempts, the dropdown menu was far too small, making the text unreadable and the widget relatively useless. I solved this issue via independent research, drawing insight from various sources. However, the most directly relevant source, and the one that finally led me to my chosen solution, was this StackOverflow question:
  + https://stackoverflow.com/questions/63770087/how-to-place-dropdowns-side-by-side-in-dash-python

## Roadmap/Features

As indicated, the current iteration of the application provides a functional and flexible web-based GUI, and supports the core essentials of CRUD functionality. However, there is room for improvement. The database and user credentials are currently hardcoded into the application, which represents a limitation in possible functionality. Additionally, filter options are limited to just three preset filters. While these accurately reflect the requested filters specified by the client, an ideal application would support arbitrary user-defined filters, in addition to hardcoded presets.

With these areas of improvement in mind, future updates could add support for interacting with multiple databases, more sophisticated user authorization and authentication, and improvements to the user interface including more dynamic filtering options.

## Contact

Primary Developer: Dylan Oldenburg - dylan.oldenburg@snhu.edu